Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3745 — Testing
2022 - 2

Tarea 1

Fecha de entrega: Miércoles 24 de Agosto del 2022 a las 23:59

Información General

La siguiente tarea contempla como objetivo principal la familiarización con los conceptos de parsing, árbol de sintaxis abstracta, el patrón composite, y el patrón visitor. Recordar que existe un foro en canvas donde el equipo de ayudantes estará atento para resolver dudas con respecto a la tarea.

Objetivos

- Comprender los conceptos de parsing y arbol de sintaxis abstracta.
- Familiarizarse con el patrón visitor y composite.
- Crear tu primer analizador de código estático.

Contexto

Considere el siguiente modelo de clases:

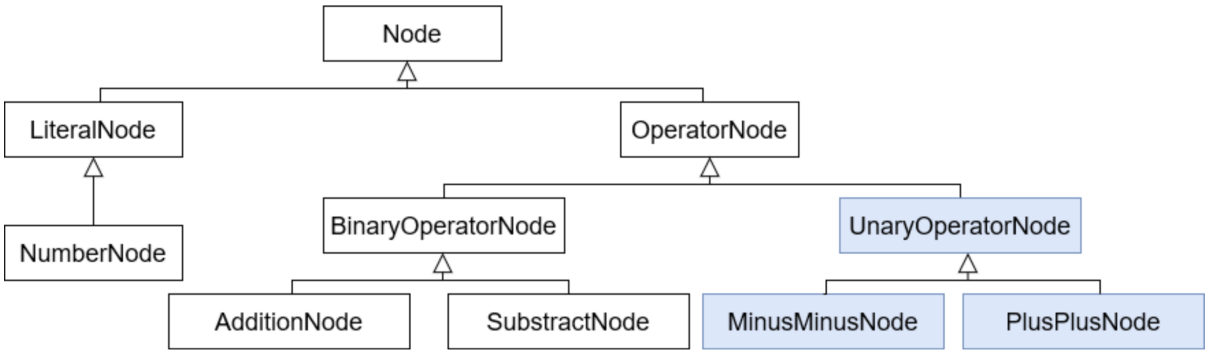

Figura 1: Diagrama Clases

Los nodos en color blanco son las clases implementadas en el curso con el profesor. Estas clases permiten modelar en forma de un árbol la componentes de un programa de sintaxis sencilla que solo soportaba operaciones aritméticas. La gramática actual está denotada en backus normal form (BNF) de la siguiente manera.

```
<expr> =  <number>
                |  (+ <expr> <expr>)
                |  (- <expr> <expr>)
```

Esto permite formas las siguientes expresiones:

```
(+ 1 1)
(+ (- 2 1) (+ 2 2))
entre otras posibles combinaciones
```

En esta tarea se quiere agregar operaciones unarias al lenguaje como ++ y - -. Después de agregar estas operaciones, el lenguaje podría soportar las siguientes expresiones:

```
(++ 1)
(+ (++ 1) (-- 2))
(+ 2 (++ 1))
entre otras posibles combinaciones
```

Y la gramática quedará de la siguiente forma:

```
<expr> =  <number>
                |  (+ <expr> <expr>)
                |  (- <expr> <expr>)
                |  (++ <expr>)
                |  (-- <expr>)
```

Tarea

Desarrolle las siguientes actividades:

- ***Actividad 1*** – Cree las clases `UnaryOperatorNode`, `PlusPlusNode`, `MinusMinusNode`, e implemente los métodos `__init__`, `__eq__`, `eval` y `to_string` en cada una de ellas.
- ***Actividad 2*** – Modifique el parser creado en clase para que pueda parsear expresiones para que soporte operadores tanto unarios (los recién agregados) y binarios (los creados en clase).
- ***Actividad 3*** – Modifique la clase `Visitor`, y agregue los métodos `visit_PlusPlus` y `visit_MinusMinus`. Además, agregue el método `accept(self, visitor)` en las clases creadas en la actividad 1.
- ***Actividad 4*** – Agregue una nueva clase llamada `UnaryOperatorCounter`, la misma debe heredar de la clase `Visitor`. Esta clase debe visitar todos los nodos del árbol y contar el número de operadores unarios presentes en una expresión, tanto `++` y `--`.

Nota: Si bien la tarea viene con algunos test de referencia, usted puede agregar mas tests para asegurar que su implementación funcione como espera. Por ejemplo, probar diferentes tipos de expresiones.

Revisión

Considere los siguientes test de ejemplo:

```
class TestParser(unittest.TestCase):
    # test para la tarea
    def test_pp(self):
        ast1 = PlusPlusNode(NumberNode(2))
        ast2 = parser("(++ 2)")
        self.assertEqual(ast1,ast2)
    def test_pp_eval(self):
        ast = parser("(++ 2)")
        result = ast.eval()
        self.assertEqual(result,3)
    def test_mm_eval(self):
        ast = parser("(-- 3)")
        result = ast.eval()
        self.assertEqual(result,2)
    def test_mix(self):
        ast = parser("(+ (++ 1) (++ 1))")
        result = ast.eval()
        self.assertEqual(result,4)
    def test_to_string2(self):
        ast1 = PlusPlusNode(MinusMinusNode(NumberNode(2)))
        self.assertEqual(ast1.to_string(),"(++ (-- 2))")
    def test_to_string3(self):
        ast1 = AdditionNode(PlusPlusNode(NumberNode(1)),MinusMinusNode(NumberNode(2)))
        self.assertEqual(ast1.to_string(),"(+ (++ 1) (-- 2))")
    def test_unary_counter(self):
        visitor = UnaryOperatorCounter()
        ast = parser("(+ (+ (++ 1) (++ 1)) (- 2 (-- 3)))")
        ast.accept(visitor)
        self.assertEqual(visitor.total(), 3)
    def test_unary_counter2(self):
        visitor = UnaryOperatorCounter()
        ast = parser("(++ 1)")
        ast.accept(visitor)
        self.assertEqual(visitor.total(), 1)
```

Los ayudantes tienen una batería de pruebas, donde se evalúan los métodos solicitados con diferentes tipos de expresiones. La nota se asignará con respecto al número de test que pasen.

- **(3 puntos)** Si pasan los test entregados en el enunciado y el código no está hardcodeado (diseñado solo para pasar los test). Se asignará un puntaje entre 0 y 3, dependiendo a cuantos test del enunciado pase.
- **(3 puntos)** Posteriormente, se ejecutarán un conjunto de tests adicionales creado por los ayudantes. Se asignará un puntaje entre 0 y 3, dependiendo de cuantos tests adicionales pase.

Archivos iniciales

En canvas se encuentra el código desarrollado por el profesor en clase con los siguientes archivos:

- *parser.py* – que contiene el parser.
- *model.py* – que contiene las clases correspondientes al arbol de sintaxis abstracta.
- *metrics.py* – que contiene los visitors creados cada uno computa una metrica.
- *test-clase.py* – que contiene el código de prueba desarrollado en clase.
- *test-ejemplo-tarea.py* – que contiene las pruebas de ejemplo del enunciado de la tarea.

Para la tarea usted debe modificar o agregar código a los archivos anteriormente mencionados segun corresponda.

Reportar problemas en el equipo

En el caso de que algún integrante no aportara como fue esperado en la tarea, podrán reportarlo enviando un correo con asunto **Problema Equipo {NumeroGrupo} Testing** a *maggie.munoz@uc.cl* y a *juanandresarriagada@uc.cl* con copia a *juanpablo.sandoval@uc.cl* explicando en detalle lo ocurrido. Posterior a eso se revisara el caso en detalle con los involucrados y se analizara si corresponde aplicar algún descuento. Instamos a todas las parejas que mantengan una buena comunicación y sean responsables con el resto de su equipo para evitar problemas de este estilo.

Restricciones y alcances

- Su programa debe ser desarrollado en **Python 3.8 o superior**.
- Los archivos entregados deben terminar con la extensión **.py**.
- En caso de dudas con respecto al enunciado deben realizarlas en un foro relacionado a la tarea que se encontrara disponible en canvas.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería de Python adicional a las utilizadas en el código base se encuentran prohibidas. En caso de que estimes necesario podrás preguntar en el foro de la tarea por el uso de alguna librería adicional.

Entrega

Código: Deberán entregar todos los archivos iniciales a excepción de los archivos de tests. La entrega se realizara por medio de un buzón de canvas habilitado para esta tarea.
Atraso: Se efectuara un descuento por entregar tareas atrasadas. Se descontara 0.5 si la tarea se entrega con menos de una hora de retraso. El puntaje final en caso de atraso seria calculado mediante la siguiente formula:

$$PuntajeFinal = PuntajeObtenido - (0,5 + 0,05 \cdot k)$$

, donde k es el numero de horas de retraso menos uno.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** o en **los grupos asignados** según sea definido en la evaluación y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

¡Éxito! :)